Strict Upward Planar Grid Drawings of Binary Trees with Minimal Area

Maarten Löffler
Utrecht University, The Netherlands

Abstract

Given a rooted binary tree T and a tuple (w, h), we wish to test whether there exists a strict upward drawing of T on a $w \times h$ grid; that is, a planar grid drawing with straight-line edges where every vertex has a strictly lower y-coordinate than its parent.

Biedl and Mondal [2] prove that this problem is NP-hard for general trees; their construction crucially uses nodes of very high degree. Akatiya et al [1] prove that the problem is also NP-hard for binary trees with a fixed combinatorial embedding; their construction crucially relies on the fixed embedding. Both pose the question for binary trees and a free embedding as an open problem.

Here, we show that this problem is also NP-hard.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Upward drawings, grid drawings, minimal area

Digital Object Identifier 10.4230/LIPIcs.GD.2024.47

Category Poster Abstract

1 Result

▶ **Theorem 1.** *Testing whether a rooted binary tree T admits a strict upward planar embedding on a $w \times h$ grid is NP-complete.*

We reduce from *monotone not-all-equal-3SAT*, which is NP-hard by Schaefer's dichotomy theorem [3]. In this problem, we are given a n variables and m clauses (triples of variables, since negative literals do not occur), and we need to find a variable assignment such that all variables in each clause are neither all **true** nor all **false**.

Given a 3SAT formula, we set $w = n + 4$ and $h = n + 4m + 1$ and construct a tree T as follows.

Variables

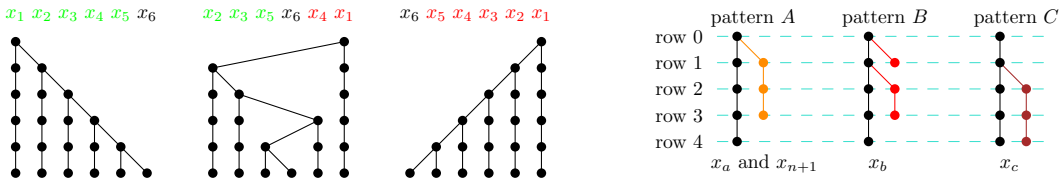

Figure 1 (left) Three embeddings of the permutation gadget. (right) The three clause patterns.

Variables are represented by paths in T of length h. In a strict upward drawing, these paths must have one vertex on every row of the grid.

The top $n + 1$ rows contain the **permutation gadget**. In the ith row, we can choose to place the path representing x_i either on the left or on the right, which encodes the **true** and **false** states of x_i. We also add a **dummy variable** x_{n+1}, which has no truth assignment. Refer to Figure 1 (left).

The vertices in the $n+1$st row, and thus the paths hanging from them, have 2^n possible permutations. Specifically, these permutations have all **true** variables to the left with increasing indices, and all **false** variables to the right with decreasing indices; these groups are separated by x_{n+1}.

Clauses

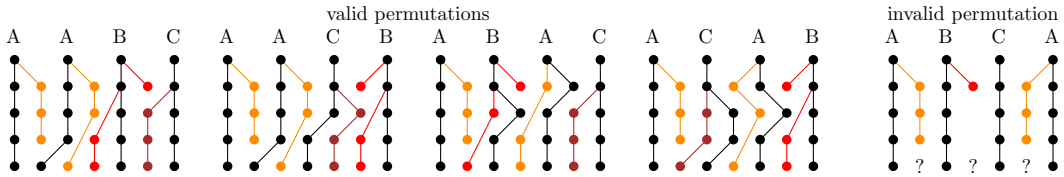

Figure 2 The clause gadget can be embedded if not both outermost paths use pattern A.

Each clause is represented by four rows, which we index 1—4; we also refer to the last row of the previous gadget as row 0. A clause (x_a, x_b, x_c) with $a < b < c$ is encoded using the four paths for x_a, x_b, x_c, and x_{n+1}. Each of these has 3 additional vertices in one or two subtrees, for 12 additional vertices in total. The total width of the construction is $n+4$; there are three "empty columns", giving 12 empty spots in these four rows. Our gadget will allow us to fill exactly these spots if and only if the clause is satisfied. We use 3 different patterns for attaching the additional vertices, where we use the same pattern for both x_a and x_{n+1}. Refer to Figure 1 (right).

▶ **Lemma 2.** *There is a valid embedding if and only if the two outermost patterns are not both A.*

Proof sketch. For the "if" part, refer to Figure 2 for example valid embeddings. For the "only if" part, when both outermost paths use pattern A, then both of them must fill rows 1, 2, and 3 of the gadget, which leaves three empty spots in row 4 but only two remaining paths, which is impossible. ◀

Note that the variable assignment in which x_a, x_b, and x_c are all **true** results in the permutaton x_a, x_b, x_c, x_{n+1}, and the assignment in which they are all **false** results in the symmetric permutaton x_{n+1}, x_c, x_b, x_a, both of which have x_a and x_{n+1} on the outside, and all other variable assignments will not have x_{n+1} on the outside. Therefore, by Lemma 2 we encode exactly the not-all-equal property of a clause.

Full construction

Our full construction (refer to accompanying poster for an example) uses straight paths without additional vertices for each variable that is not part of a clause. The correctness relies the ability of these paths to not influence the gadget.

▶ **Lemma 3.** *The presence or absence of any vertical paths separating the four subtrees in a clause gadget has no influence on the satisfiability of the gadget.*

We also need the clause gadgets to be independent; that is, additional vertices in one clause gadget should not fill empty spots in different gadgets.

▶ **Lemma 4.** *The entire construction can be embedded if and only if each clause can be individually satisfied.*

2 Future Work

Our construction relies critically on the strictness of the drawings. What is the complexity of finding **non-strict** upward planar embeddings of trees on a given grid?

References

1 Hugo Akitaya, Maarten Löffler, and Irene Parada. How to fit a tree in a box. *Graphs and Combinatorics*, 2022.

2 Therese Biedl and Debajyoti Mondal. On upward drawings of trees on a given grid. In Fabrizio Frati and Kwan-Liu Ma, editors, *Graph Drawing and Network Visualization*, pages 318–325, Cham, 2018. Springer International Publishing.

3 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804350.